Documento de Diseño: Generación de gráficos y texto

1st Daniel Duarte Cordero
Instituto Tecnológico de Costa Rica
Ingeniería en Computadores
Cartago, Costa Rica
daduarte@estudiantec.cr

I. Listado de requerimientos del sistema

A. Funcionamiento

- El sistema debe permitir al usuario seleccionar un cuadrante específico (del 1 al 16) de una imagen de entrada.
- El procesamiento de la imagen debe realizarse únicamente sobre el cuadrante seleccionado.
- El algoritmo de interpolación bilineal debe ejecutarse completamente en lenguaje ensamblador.
- La imagen resultante debe incluir el cuadrante interpolado, manteniendo los demás cuadrantes sin modificar.
- El resultado debe mantenerse coherente en cuanto a valores de intensidad, generando transiciones suaves entre píxeles.
- El resultado debe mantenerse coherente en cuanto a valores de intensidad, generando transiciones suaves entre píxeles.
- La imagen interpolada debe ser exportada a un archivo .img para ser visualizado posteriormente.

B. Entradas

- Imagen de entrada en escala de grises, con valores de píxeles entre 0 y 255.
- Dimensiones mínimas de la imagen: 390×390 píxeles.
- Archivo .img ubicado en el mismo directorio que el ejecutable del programa.
- Valor numérico entre 1 y 16 que indica el cuadrante seleccionado por el usuario.

C. Procesamiento

- Aplicación del algoritmo de interpolación bilineal usando enteros y ponderaciones basadas en cercanía.
- Lectura y escritura de imágenes desde y hacia archivos binarios.
- Cálculo de nuevos valores de píxeles intermedios a partir de píxeles conocidos.
- Automatización del flujo completo: desde lectura hasta generación del resultado final.
- El procesamiento debe realizarse utilizando un simulador de arquitectura (ARM, x86 o RISC-V).

D. Salida

- Generación de una nueva imagen .img con el cuadrante seleccionado ya interpolado.
- Visualización del proceso mediante un programa en alto nivel, que muestre:
  - Imagen original.
  - Cuadrante seleccionado
  - Resultado final de la interpolación.

E. Herramientas

- Simulador del ISA (Instruction Set Architecture) elegido: ARM, x86 o RISC-V.
- Programa de alto nivel para visualización (Python, Octave, Matlab u otro).
- Editor de código o entorno de desarrollo compatible con ensamblador.
- Generador de funciones como entrada de señales.

F. Normativas y estándares

Para el desarrollo de este proyecto es necesario adherirse a ciertas normativas y estándares reconocidos que aseguran la calidad y precisión.

1) Estándares de representación y procesamiento de imágenes:

a) OpenCV Image Standards: [12]

- Las imágenes en escala de grises deben almacenarse como matrices uint8 con valores entre 0 y 255.
- El acceso a píxeles se realiza en formato (fila, columna), es decir, (y, x) para evitar confusión en visualización.
- El formato más básico compatible es una matriz plana sin encabezado, si las dimensiones son conocidas de antemano.
- Las imágenes deben almacenarse en orden fila a fila, igual que en JPEG y otros estándares.
- Para asegurar compatibilidad, se recomienda usar dimensiones múltiples de 2 o 4, lo cual evita errores en procesos de visualización e interpolación.

2) Estándares en desarrollo de software:

a) IEEE Std 830-1998: [6]

- Todo proyecto debe incluir un documento formal de requerimientos del sistema, claramente estructurado y verificable.

- Cada requerimiento debe ser verificable, es decir, que pueda ser probado o evaluado.
- Se deben evitar términos como ”rápido”, ”fácil”, ”eficiente” sin una métrica concreta.
- Todo requerimiento debe estar numerado o identificado para facilitar trazabilidad.
- Los requerimientos deben ser modificables: el documento debe poder actualizarse sin pérdida de consistencia.

3) Normativas y Estándares en Arquitectura de Computadoras y Ensamblador: En el desarrollo de este proyecto se procurará seguir las buenas prácticas reconocidas internacionalmente para la programación en lenguaje ensamblador y el diseño de software en bajo nivel, tomando como referencia general las especificaciones de arquitecturas ampliamente aceptadas como RISC-V, ARM y x86-64. Las siguientes directrices generales son adoptadas: [1], [14], [15]

- Respetar el modelo de programación definido por el conjunto de instrucciones de la arquitectura seleccionada (alineación, registros, codificación binaria, direccionamiento).
- Evitar instrucciones dependientes de extensiones propietarias o no portables, privilegiando instrucciones del conjunto base.
- Escribir código modular, legible y documentado, separando secciones de entrada, procesamiento e interacción con memoria.
- Utilizar registros de forma eficiente, evitando sobrescritura innecesaria o conflictos en contextos de subrutinas.
- Seguir prácticas de alineación de datos y acceso secuencial para mejorar el rendimiento en ejecución simulada.
- Garantizar la compatibilidad del programa con simuladores ISA comunes, respetando los formatos de entrada/salida definidos.

4) Buenas prácticas en control de versiones:

a) GitFlow (nvie.com): [5] Consiste en una estrategia de control de versiones que organiza el repositorio en dos ramas principales, master y develop. En el caso de la rama master esta contiene aquellas versiones estables y que se encuentran listas para producción. Por otra parte, develop es utilizada para desarrollar nuevos features o correciones, esto a partir de subramas temporales de develop. Cuano se terminan de desarrollar las funcionalidades de dichas subramas se hace un merge al develop, para posteriormente realizar un merge a la master. En este punto es donde se genera una etiqueta (tag), que tiene la función de representar la versión lanzada. Esta estragia es clave para facilitar la organización, así como la trazabilidad de cambios, permitiendo una colaboración ordenada en el desarrollo.

b) Conventional Commits: [3] Esta conveción se enfoca en los mensajes de commit en Git. Establece que cada commit debe tener un prefijo que indique el tipo de cambio realizado. Como lo es el ”feat”, cuando se desarrolla una nueva funcionalidad, así como ”fix” para la correción de errores, docs para cambios en la documentación o ”refactor” al refactorizar el código. La práctica mencionada permite mejorar la legilibidad del historial de cambios, además de que contribuye a generar automáticamente los registros de cambios(changelogs) en el proyecto.

5) Normas de calidad y documentación:

a) ISO/IEC 9126 / 25010: [8], [9] Estos estándares definen un conjunto de características necesarias para la evaluación de la calidad del producto, haciendo énfasis en la funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad. En donde se busca que el software cumpla con los requisitos planteados, así como que sea confiable, apegado a diferentes condiciones de uso, sea fácil de entender y utilizar, se procure usar los recursos disponibles de manera eficiente. Además que esté abierto a modificaciones con el objetivo de ser adaptado a distintos entornos de ejecución en caso de que sea necesario.

b) Markdown y README.md: [10] El lenguaje de mercado Markdown se toma como guía para este proyecto. Primeramente, se elabora un archivo principal README.md, el cual describe de manera general el propósito del proyecto, así como las instrucciones de instalación y ejecución, requisitos de dependencias (en caso de que se presenten), estructura de directorios y cualquier otra información relevante. El uso de Markdown garantiza que la documentación sea fácilmente accesible, editable y visualizable para diferentes plataformas.

G. Estado del arte

1) Soluciones similares: En el área de la interpolación de imágenes y procesamiento de datos de bajo nivel, se han desarrollado diferentes soluciones y metologías que abordan problemas simialares a este proyecto.

Por ejemplo, la interpolación bilineal es una técnica bastante conocida por su escalado de imágenes. Razón por las que han surgido bibliotecas de procesamiento de imágnes como OpenCV y PIL. Estas herramientas proporcionan funciones ya optimizadas para aplicar interpolaciones a imágenes de alta resolución[13]. Usualmente las implementaciones se desarrollan en lenguajes de alto nivel. Aunque internamente aprovechan extensiones de hardware como SSE y AVX para acelerar el procesamiento [7].

En implementaciones con el lenguaje ensamblador se han desarrollado soluciones que optimizan las operaciones de procesamiento de imágenes, enfocándose en aquellas aplicaciones en tiempo real, como lo son la compresión de video o el procesamiento de transmisiones multimedia [2]. Al usar ensamblador se le saca provecho a la cercanía que se tiene con el hardware, con el fin de optimizar la manipulación de matrices de píxeles y así reducir latencia y mejora la eficencia energética.

2) Proyectos similares:

- Primeramente se tiene el proyecto ”Image Resampling with SIMD Instructions”, el proyecto fue desarrollado por Intel, en el mismo se acude principalmente a las instrucciones SIMD, empleadas para acelerar operaciones de redimensionamiento de imágenes, entre las técnincas utilizadas se encuentra la interpolación bilineal. Si bien, en su mayor parte el proyecto se implementa en C++, se presentan ciertas partes en ensamblador optimizado para x86, las cuales muestran técnincas de alineación de datos, uso de registros vectoriales y optimización del acceso a memoria. [7].

- También se tiene como referencia al proyecto ”Bilinear Interpolation on FPGA Hardware”, este fue desarrollado en el curso de Sistemas Computacionales del MIT, en el cual se realizó la interpolación bilineal directamente con hardware usando una FPGA. En este trabajo se hace énfasis a los aspectos de bajo nivel, precisamente en el manejo de datos binarios, control de flujos de datos y la optimización de recusos de hardware [11]. Los principios del proyecto mencionado son aplicables al desarrollo de este proyecto, con la salvedad que se trabajan con diferentes niveles de abstracción, pero siendo sumamnte útil para procurar un diseño eficiente de algoritmos de ensamblador.
- El proyecto ”Real-Time Processing in Embedded Systems” describe el diseño e implementación de sistemas de procesamiento de imágenes en tiempo real utilizando microconttoladores programados en C y ensamblador. El mismo analiza técnicas para optimizar la manipulación de píxeles, minimizar la latencia y el gestonamiento del accedo a la memoria eficientemente. Estos últimos principios son una guía que pueden resultar bastante provechoso para aplicaciones en el desarrollo de soluciones de diferentes arquitecturas en el procesamiento de imágenes.
- Finalmente se toma en cuenta el proyecto ”Optimized Image Processing Using Assembly Languaje”, el cual corresponde a un artículo que plantea una investigación en la que se implementan algoritmos de procesamiento de imágenes (incluyendo interpolación), esto implementado completamente en ensamblador. Se destaca la importancia en la optimización de carga y almacemaniento de píxeles, el uso de instrucciones SIMD y la alineción de memoria para maximizar el rendimiento del procesamiento[2].

II. Evaluación de opciones de solución al problema

Para evaluar las posibles soluciones al problema es necesario contemplar por separado diferentes secciones con sus respectivas opciones.

A. Lenguaje de programación de alto nivel

- **Python** : Es un lenguaje de programación interpretado, reconocido por su sintaxis sencilla y su rapidez de desarrollo. Posee una extensa colección de bibliotecas de procesamiento de imágenes, como OpenCV y Pillow, que permiten la carga, manipulación y visualización de archivos de imagen de forma eficiente.
- **C++**: Lenguaje compilado que proporciona un control preciso sobre los recursos del sistema, como la memoria y los procesos de entrada/salida. Se utiliza ampliamente en aplicaciones donde el rendimiento es crítico. Existen múltiples bibliotecas optimizadas para C++ que ofrecen funcionalidades específicas de procesamiento de imágenes.
- **Java**: Lenguaje de programación orientado a objetos que se caracteriza por su portabilidad entre diferentes plataformas mediante el uso de la máquina virtual Java (JVM). Cuenta con bibliotecas que permiten trabajar con archivos gráficos y realizar operaciones de procesamiento básico de imágenes.

B. Visualización y validación de la imagen interpolada

- **Visualización automática mediante script**: Consiste en utilizar un programa automatizado, escrito en un lenguaje de alto nivel como Python o Matlab, que se encargue de abrir el archivo de imagen generado y mostrarlo en pantalla de manera directa, facilitando la verificación de los resultados.
- **Visualización manual**: Implica abrir el archivo de salida utilizando una aplicación de visualización estándar, como un visor de imágenes convencional. En este enfoque, el usuario carga manualmente la imagen para inspeccionar los resultados obtenidos tras el proceso de interpolación.

C. Sistema operativo

- **Linux**: Sistema operativo de código abierto basado en Unix, utilizado ampliamente en entornos académicos e industriales. Ofrece una gran variedad de herramientas de desarrollo y soporte para programación en bajo nivel.
- **Windows**: Sistema operativo comercial ampliamente difundido en entornos de usuario general y profesional. Presenta compatibilidad con diversas herramientas de desarrollo y una amplia disponibilidad de software.
- **macOs**: Sistema operativo desarrollado por Apple, también basado en Unix. Se distingue por su estabilidad, facilidad de uso y disponibilidad de herramientas de desarrollo específicas.

D. Arquitectura

- **ARM**: Arquitectura basada en un conjunto reducido de instrucciones (RISC), diseñada para lograr alta eficiencia energética y bajo consumo de recursos. Es ampliamente utilizada en dispositivos móviles, sistemas embebidos y plataformas donde la eficiencia es prioritaria.
- **RISC-V** Arquitectura basada en el modelo RISC, de diseño abierto y extensible. Su naturaleza modular permite adaptaciones específicas para diferentes aplicaciones. RISC-V está ganando popularidad en el ámbito académico e industrial por su flexibilidad y costo reducido.
- **x86** Arquitectura de conjunto complejo de instrucciones (CISC), tradicionalmente empleada en computadoras de escritorio, servidores y laptops. Ofrece un amplio conjunto de instrucciones, madurez tecnológica y una vasta disponibilidad de herramientas de desarrollo.

E. Forma de carga de la imagen de entrada

- **Carga completa en memoria**: Se realiza la lectura completa de la imagen desde el almacenamiento al inicio de la ejecución, almacenándola en la memoria principal (RAM) para su posterior procesamiento.
- **Carga línea por línea**: Consiste en leer la imagen de manera parcial, procesándola por bloques o líneas, lo que permite manejar imágenes de gran tamaño sin necesidad de cargarla completamente en memoria.

F. Recorrido de píxeles de entrada

- **Con solapamiento**: Durante el recorrido de los píxeles de la imagen, se aprovechan los valores de píxeles adyacentes que han sido procesados previamente para calcular nuevos resultados, reutilizando la información existente..
- **Sin solapamiento**: Cada sección de la imagen es tratada de forma independiente, sin considerar valores previamente procesados, lo que implica un flujo de procesamiento completamente separado entre distintas regiones.

G. Almacenamiento de píxeles interpolados

- **Escribir en el .img en cada iteración**: A medida que se calculan los valores interpolados de los píxeles, estos son escritos directamente en el archivo de imagen de salida, en una operación secuencial.
- **Uso de buffer del tamaño de la imagen interpolada**: Se reserva un área de memoria para almacenar todos los píxeles interpolados conforme se van generando, de modo que el archivo final se escribe en una única operación al concluir el procesamiento.

H. Tipo de direccionamiento en ensamblador para acceder a píxeles

- **Direccionamiento directo**: Se realiza el acceso a los datos de la imagen utilizando direcciones fijas o predefinidas, aplicando desplazamientos constantes calculados de antemano.
- **Direccionamiento indexado**: Se calcula la posición de memoria de cada píxel de manera dinámica utilizando registros como índices y realizando operaciones aritméticas en tiempo de ejecución.

I. Optimización en uso de registros

- **Uso mínimo de registros**: La estrategia consiste en utilizar solo los registros necesarios para el procesamiento, almacenando los resultados intermedios en memoria de manera frecuente para mantener el código simple.
- **Uso intensivo de registros**: Se procura mantener la mayor cantidad posible de datos en los registros del procesador, minimizando el número de accesos a memoria durante el procesamiento.
- **Uso combinado de registros y memoria**: Se adopta un enfoque equilibrado donde los datos más utilizados o críticos se almacenan en registros, mientras que otros datos se mantienen en memoria de forma temporal.

III. Comparación de opciones de solución

Una vez identificadas las alternativas disponibles para cada uno de los aspectos considerados, se procede a realizar una comparación general de las mismas. Este análisis tiene como objetivo identificar las principales fortalezas y limitaciones de cada opción, considerando factores de rendimiento, facilidad de implementación, adaptabilidad y disponibilidad de herramientas.

A. Lenguaje de programación de alto nivel

Python ofrece una curva de aprendizaje reducida y una rápida implementación de soluciones, lo que facilita el desarrollo de scripts de apoyo para la visualización y el procesamiento de imágenes. Sin embargo, su desempeño en operaciones de cómputo intensivo puede ser limitado en comparación con lenguajes compilados.

C++ proporciona un control exhaustivo sobre los recursos del sistema y permite alcanzar un rendimiento superior, especialmente en tareas que requieren operaciones matemáticas intensivas. No obstante, la complejidad asociada a la gestión manual de memoria y a la programación de bajo nivel puede incrementar significativamente el tiempo de desarrollo.

Java, aunque es altamente portable y sencillo de utilizar en comparación con C++, no es la opción más eficiente para tareas de procesamiento de imágenes en bajo nivel. Su modelo de ejecución sobre una máquina virtual introduce una capa de abstracción que limita el acceso directo al hardware.

B. Visualización y validación de la imagen interpolada

La visualización automática mediante scripts permite agilizar la validación de resultados, integrar pruebas de forma sistemática y reducir el margen de error humano. No obstante, requiere invertir tiempo en la creación y mantenimiento de dichos scripts.

La visualización manual, aunque es sencilla de implementar y no depende de programación adicional, resulta menos eficiente en escenarios donde es necesario validar múltiples imágenes o realizar comprobaciones sistemáticas.

C. Sistema operativo

Linux ofrece un entorno favorable para el desarrollo de aplicaciones que requieren acceso a bajo nivel, con soporte nativo para herramientas de ensamblador y manipulación directa de archivos binarios. Sin embargo, su adopción puede requerir una curva de aprendizaje para usuarios acostumbrados a entornos más gráficos.

Windows es ampliamente utilizado y compatible con una vasta gama de software de desarrollo, pero ciertas operaciones de bajo nivel pueden ser menos accesibles o requerir herramientas adicionales para su correcta ejecución.

macOS, al compartir su base Unix con Linux, proporciona estabilidad y herramientas robustas, aunque su ecosistema cerrado puede limitar el acceso a configuraciones avanzadas y su dependencia del hardware Apple puede representar una restricción.

D. Arquitectura

La arquitectura ARM es reconocida por su alta eficiencia energética y su bajo consumo de potencia, características que la han convertido en el estándar en dispositivos móviles y sistemas embebidos. Gracias a su simplicidad y diseño RISC, ARM facilita la implementación de soluciones optimizadas para entornos donde los recursos son limitados. Sin embargo, en términos de herramientas de desarrollo y soporte para programación de bajo nivel en plataformas de escritorio,

ARM puede presentar algunas restricciones adicionales en comparación con arquitecturas más consolidadas.

RISC-V, al ser una arquitectura abierta, permite una personalización profunda del conjunto de instrucciones, lo cual la hace especialmente atractiva para proyectos de investigación, innovación y desarrollo de hardware especializado. Su diseño modular ofrece flexibilidad y adaptación a distintos escenarios. No obstante, su adopción todavía está en crecimiento, por lo que algunas herramientas de desarrollo y documentación pueden ser menos maduras o limitadas en comparación con alternativas más establecidas.

La arquitectura x86, ampliamente adoptada en computadoras personales y servidores, ofrece una madurez tecnológica considerable, una disponibilidad extensa de herramientas de desarrollo y una robusta compatibilidad de software. Su conjunto de instrucciones más amplio y su soporte de largo plazo la hacen ideal para aplicaciones que requieren procesamiento intensivo y entornos de desarrollo bien establecidos. Sin embargo, su diseño CISC implica una mayor complejidad en la ejecución de instrucciones a nivel de hardware.

E. Forma de carga de la imagen de entrada

La carga completa de la imagen en memoria facilita el acceso inmediato a cualquier píxel, permitiendo una programación más directa y menos propensa a errores de lectura. Sin embargo, esta estrategia depende de la disponibilidad de suficiente memoria RAM, especialmente cuando se trabaja con imágenes de alta resolución.

La carga línea por línea optimiza el consumo de memoria, permitiendo trabajar con imágenes de gran tamaño en sistemas con recursos limitados. A cambio, introduce mayor complejidad en el manejo de los datos, especialmente cuando es necesario acceder a píxeles vecinos para operaciones como la interpolación bilineal.

F. Recorrido de píxeles de entrada

El recorrido con solapamiento aprovecha resultados intermedios, lo que puede mejorar la eficiencia del procesamiento al reducir la cantidad total de operaciones realizadas. Esta estrategia, sin embargo, incrementa la complejidad en el control de flujos de datos y puede dificultar la depuración.

El recorrido sin solapamiento simplifica la estructura del procesamiento, permitiendo un flujo de trabajo más lineal y sencillo de implementar. No obstante, puede resultar en un mayor número de cálculos redundantes y, en consecuencia, en un uso menos eficiente de los recursos.

G. Almacenamiento de píxeles interpolados

La escritura de píxeles en el archivo de imagen en cada iteración permite conservar el consumo de memoria al mínimo, dado que no se requiere almacenamiento intermedio. Sin embargo, las operaciones de escritura frecuentes pueden afectar negativamente el rendimiento general del procesamiento.

El uso de un buffer de memoria para almacenar todos los píxeles antes de escribirlos en el archivo final ofrece un procesamiento más eficiente, al reducir la interacción con el sistema de archivos. Esta estrategia, sin embargo, implica la necesidad de reservar espacio adicional en la memoria del sistema.

H. Tipo de direccionamiento en ensamblador para acceder a píxeles

El direccionamiento directo simplifica el acceso a los datos cuando las posiciones de memoria son fijas y conocidas, reduciendo la complejidad del código ensamblador. No obstante, limita la flexibilidad, ya que cambios en las dimensiones de la imagen requerirían ajustes manuales en los cálculos de direcciones.

El direccionamiento indexado, por su parte, permite adaptar el procesamiento a imágenes de diferentes tamaños y estructuras de datos dinámicas. Esta mayor flexibilidad viene acompañada de una mayor complejidad en el código, debido a la necesidad de calcular desplazamientos dinámicos durante la ejecución.

I. Optimización en uso de registros

El uso mínimo de registros simplifica la programación y reduce la necesidad de gestionar explícitamente los recursos internos del procesador. Sin embargo, esta estrategia puede conducir a un mayor número de accesos a memoria, afectando negativamente el tiempo de ejecución.

El uso intensivo de registros mejora el rendimiento del programa al minimizar los accesos a memoria, aprovechando la velocidad de acceso de los registros. Este enfoque requiere una gestión más cuidadosa para evitar conflictos y sobrecargas.

La estrategia de uso combinado de registros y memoria ofrece un equilibrio entre rendimiento y simplicidad, permitiendo optimizar el acceso a los datos más críticos mientras se conserva una estructura de código relativamente sencilla y manejable.

IV. Selección de la propuesta final

Luego de realizar la evaluación y comparación de las diferentes alternativas disponibles para cada aspecto relevante del proyecto, en esta sección se procede a seleccionar las opciones que mejor se adaptan a los requerimientos establecidos y a las características específicas de la solución a implementar. La selección se fundamenta en criterios técnicos, considerando factores como la eficiencia, la facilidad de implementación, la disponibilidad de herramientas, la compatibilidad con el entorno de trabajo y la optimización de recursos.

Cada decisión tomada se justifica de manera objetiva, asegurando la coherencia entre las alternativas elegidas y los objetivos del proyecto, con el fin de garantizar una implementación efectiva y un desempeño óptimo del sistema desarrollado.

A. Justificación de la solución seleccionada

1) Lenguaje de programación de alto nivel seleccionado: Python: Para este proyecto, se seleccionó Python como lenguaje de alto nivel debido a su capacidad para satisfacer de manera eficiente los requerimientos específicos de conversión

de imágenes, automatización de procesos y visualización de resultados.

En primer lugar, Python permitió realizar de forma sencilla la conversión de imágenes originales en formato .jpg a imágenes en escala de grises, que son requeridas por el programa de ensamblador para realizar la interpolación. Utilizando la librería Pillow (PIL), se implementó la conversión de imágenes a blanco y negro (convert('L')), así como la redimensión de la imagen original a las dimensiones específicas solicitadas (390x390 píxeles).

Posteriormente, mediante NumPy, fue posible transformar la imagen en una matriz de valores numéricos y extraer cualquier cuadrante en formato crudo (.img). Este archivo crudo sirve de entrada directa al programa en ensamblador, cumpliendo con el requisito de trabajar únicamente sobre archivos de tipo binario de 8 bits.

Además, Python facilita el control automatizado de todo el flujo de trabajo mediante el módulo subprocess, permitiendo ensamblar (nasm), ligar (ld) y ejecutar el programa ensamblador directamente desde el entorno de Python sin intervención manual. Esto asegura que cada vez que el usuario seleccione un cuadrante y solicite interpolarlo, el procesamiento completo sea ejecutado de forma continua y automática, cumpliendo con la necesidad de mantener separado el procesamiento en ensamblador del entorno de alto nivel, pero coordinándolos de manera fluida.

El lenguaje también permitió construir una interfaz gráfica amigable utilizando Tkinter. A través de esta interfaz, el usuario puede visualizar la imagen completa dividida en cuadrantes, seleccionar el cuadrante deseado, iniciar la interpolación, y observar tanto el resultado sin interpolar como la imagen interpolada generada por el ensamblador. La integración con ImageTk (de Pillow) facilitó la visualización directa de las imágenes procesadas dentro de la interfaz.

En resumen, Python fue seleccionado porque resolvió de forma integral las necesidades del proyecto, proporcionando:

- Herramientas simples y potentes para leer, procesar y convertir imágenes.
- Capacidad de automatizar la ejecución del programa de ensamblador, ensamblaje y generación de resultados.
- Facilidades para crear una interfaz gráfica intuitiva para la selección de cuadrantes y visualización de resultados.
- Flujo de trabajo continuo que garantiza que la imagen interpolada provenga exclusivamente del procesamiento en ensamblador, como lo exige el enunciado del proyecto.

2) Visualización y validación de la imagen interpolada seleccionada: Visualización automática mediante script: Para la visualización y validación de la imagen interpolada, se optó por implementar un esquema de visualización automática mediante un script desarrollado en Python. Esta decisión se fundamentó en la necesidad de integrar de forma fluida y eficiente todo el proceso de selección, ejecución y validación de resultados dentro de un mismo entorno de trabajo.

Al automatizar la visualización, se logró que el usuario pudiera observar directamente el resultado del proceso de interpolación sin requerir pasos manuales adicionales, como abrir archivos de imagen externos o utilizar visores independientes. La integración de las funciones de procesamiento y visualización en una única interfaz gráfica proporciona una experiencia de usuario más consistente, reduciendo tiempos de verificación y minimizando la posibilidad de errores humanos en la inspección de los resultados.

Además, el enfoque automático permite reflejar de inmediato cualquier cambio en los datos procesados, lo cual es especialmente importante en un proyecto como este, donde la correcta interpolación de los valores de píxeles debe ser verificada de manera visual en cada ejecución. Esta dinámica facilita realizar múltiples pruebas sobre diferentes cuadrantes sin interrumpir el flujo de trabajo, permitiendo un análisis ágil y efectivo de la calidad del algoritmo de interpolación bilineal desarrollado en ensamblador.

La automatización también ayuda a cumplir con uno de los principios fundamentales del proyecto: mantener la separación entre el procesamiento en bajo nivel y la interacción en alto nivel, utilizando Python únicamente para visualizar los resultados y no para intervenir en el procesamiento de los datos.

En consecuencia, la visualización automática mediante script no solo resultó ser una opción práctica, sino también una elección estratégica que mejora la eficiencia operativa del sistema, asegura la correcta validación del procesamiento y ofrece una experiencia de usuario final mucho más ágil y profesional.

3) Sistema operativo seleccionado: Linux: En plataformas Windows, el uso de herramientas de bajo nivel para trabajar directamente con ensamblador presenta múltiples complicaciones adicionales. Es necesario instalar entornos específicos (como Cygwin, MinGW o WSL), configurar adecuadamente las rutas de compiladores y enlazadores, y en muchos casos, adaptar el código para compatibilizarlo con las llamadas al sistema específicas de Windows, las cuales difieren de las llamadas estándar en sistemas tipo Unix.

En cambio, Linux proporciona un entorno nativo y robusto para la programación en bajo nivel, donde herramientas como NASM (Netwide Assembler) para la generación de archivos objeto (.o) y LD (GNU Linker) para el enlazado están disponibles directamente en los repositorios oficiales y se instalan fácilmente mediante el gestor de paquetes. Además, las llamadas al sistema (syscalls) siguen la convención estándar de Unix/Linux, lo cual simplifica considerablemente el desarrollo del programa en ensamblador.

La integración con Python también es más directa en Linux, ya que el módulo subprocess permite invocar de manera sencilla los comandos del sistema para ensamblar, enlazar y ejecutar el programa en ensamblador, todo dentro de un mismo script. Esto asegura que el flujo de trabajo sea completamente automatizado, sin necesidad de pasos adicionales o configuraciones complicadas.

Adicionalmente, Linux ofrece un mejor control sobre los permisos de archivos, manejo de memoria y operaciones de bajo nivel, factores que son relevantes en un proyecto donde el procesamiento directo de archivos binarios (.img) y el acceso controlado a recursos del sistema son esenciales.

4) Arquitectura seleccionada: x86: Para la implementación del programa de interpolación bilineal en lenguaje ensamblador, se seleccionó la arquitectura x86. Esta decisión se

sustentó en varios factores prácticos y técnicos relevantes para el éxito del proyecto.

En primer lugar, la arquitectura x86 ofrece un conjunto amplio y maduro de instrucciones, lo que facilita la implementación de operaciones tanto aritméticas como de manipulación de memoria necesarias para el procesamiento de imágenes. La disponibilidad de instrucciones específicas para operaciones de carga, almacenamiento y cálculo de desplazamientos simplifica la construcción de algoritmos eficientes a nivel de ensamblador.

En comparación con arquitecturas como RISC-V, donde la instalación del toolchain y el entorno de desarrollo puede ser compleja y demandar una considerable cantidad de tiempo de configuración, trabajar en x86 resulta significativamente más accesible. Las herramientas necesarias para la programación en esta arquitectura, como NASM para la generación de archivos objeto y LD para el enlazado, están ampliamente disponibles en los repositorios de Linux y su configuración es directa y estable.

Por otra parte, la arquitectura x86 facilita el acceso y control eficiente tanto de los registros como de la memoria, lo cual es crucial en un proyecto como este, donde el procesamiento implica la lectura, interpolación y escritura de grandes cantidades de datos en archivos de imagen binarios. La riqueza del conjunto de registros generales y extendidos (por ejemplo, RAX, RBX, RCX, RDX, etc.) permite organizar de manera estructurada los valores de píxeles y las direcciones de memoria durante el proceso de interpolación.

Otra ventaja importante es que x86 es una arquitectura ampliamente documentada y soportada, con abundante material de referencia, ejemplos de código y soporte en comunidades técnicas. Esto garantiza un entorno de desarrollo más predecible, donde los errores pueden ser diagnosticados y corregidos más rápidamente gracias a la disponibilidad de recursos.

Por todas estas razones, la arquitectura x86 fue seleccionada, ya que ofrece una combinación de facilidad de instalación, madurez del entorno de desarrollo, flexibilidad en el manejo de registros y memoria, y una infraestructura de soporte técnico que facilita la implementación eficiente del programa de interpolación bilineal en ensamblador.

5) Forma de carga de la imagen de entrada seleccionada: Carga completa en memoria: Para la carga de la imagen de entrada, se optó por realizar una lectura completa en memoria antes de iniciar el procesamiento. Esta decisión se basó en criterios de eficiencia, simplicidad y aprovechamiento de los recursos disponibles en el equipo de trabajo.

Realizar constantes operaciones de lectura desde el archivo mediante llamadas al sistema (syscalls) implica un gasto considerable de tiempo de procesamiento, dado que cada syscall representa una transición de modo usuario a modo kernel, lo cual genera una sobrecarga que puede ser significativa al procesar un gran número de píxeles. Además, abrir y cerrar el archivo de forma reiterada incrementaría la complejidad del código, haciendo el control de errores y la gestión de recursos más propensos a fallos.

Al almacenar toda la imagen en un buffer de memoria desde el inicio, se garantiza un acceso inmediato y constante a cualquier píxel durante el proceso de interpolación, sin depender de operaciones de entrada/salida adicionales. Esta estrategia mejora el rendimiento general del programa y simplifica el flujo de procesamiento, ya que basta con operar sobre direcciones de memoria internas sin necesidad de manejar el archivo de entrada durante la interpolación.

La decisión de cargar la imagen completa en memoria es también viable gracias a las condiciones del entorno de ejecución. El tamaño de la imagen de entrada es de 97x97 píxeles, equivalente a menos de 10 KB de datos (9,409 bytes), un tamaño mínimo en comparación con la capacidad de memoria RAM disponible en una computadora portátil moderna. De esta forma, el consumo de recursos es despreciable y no representa una limitación para el sistema.

6) Recorrido de píxeles de entrada seleccionado: Con solapamiento: Para el recorrido de los píxeles de entrada durante el proceso de interpolación bilineal, se optó por implementar una estrategia de recorrido con solapamiento. Esta elección se fundamentó principalmente en la necesidad de obtener resultados más suaves y continuos en la imagen interpolada.

En un proceso de interpolación, especialmente en imágenes digitales, la continuidad y la suavidad de las transiciones entre píxeles son factores críticos que afectan la calidad visual del resultado. Al emplear una estrategia de solapamiento, se reutilizan los valores de píxeles base (A, B, C, D) en múltiples cálculos para generar los nuevos valores interpolados. Esta reutilización permite que los nuevos píxeles interpolados estén matemáticamente relacionados con sus vecinos inmediatos, lo que garantiza que los cambios de intensidad en la imagen sean graduales y coherentes.

En contraste, un recorrido sin solapamiento, donde cada región se procesa de manera independiente, puede introducir discontinuidades o variaciones abruptas, ya que los datos de entrada no se aprovechan de forma conjunta para suavizar las transiciones.

Además de la mejora visual, el recorrido con solapamiento optimiza el número de lecturas de memoria, ya que los valores base de los píxeles son cargados una sola vez y utilizados en varios cálculos posteriores, mejorando así la eficiencia del procesamiento en términos de acceso a datos.

Esta decisión también es coherente con el objetivo del proyecto de implementar un algoritmo de interpolación bilineal fiel a su definición matemática, donde cada nuevo píxel interpolado depende de combinaciones ponderadas de múltiples píxeles originales.

7) Almacenamiento de píxeles interpolados seleccionado: Uso de buffer del tamaño de la imagen interpolada: La estrategia seleccionada para almacenar los resultados de la interpolación bilineal fue el uso de un buffer en memoria que contiene toda la imagen interpolada antes de realizar la escritura al archivo de salida.

Esta decisión se basó en los mismos criterios que fundamentaron la forma de carga de la imagen de entrada: evitar realizar múltiples operaciones de entrada/salida sobre el sistema de archivos, las cuales generan una sobrecarga considerable debido al costo de las llamadas al sistema (syscalls). Al almacenar los datos en un único buffer, se minimiza el número de operaciones de escritura, permitiendo transferir todos los

datos interpolados al almacenamiento de una sola vez, de manera eficiente.

El tamaño final de la imagen resultante también influyó en esta decisión. Debido a que se implementó una interpolación bilineal con solapamiento, el número de píxeles se incrementa en ambas dimensiones. A partir de un cuadrante original de 97×97 píxeles, la imagen interpolada resultante alcanza un tamaño de 385×385 píxeles. Este aumento significativo de resolución implica la generación de una cantidad considerable de nuevos datos, que deben ser gestionados de manera ordenada en memoria antes de ser almacenados en el archivo de salida.

La disponibilidad de suficiente memoria RAM en el equipo de trabajo permitió adoptar esta estrategia sin afectar el rendimiento general del sistema, dado que el tamaño total del buffer (aproximadamente 148 KB) es trivial para una computadora portátil moderna.

8) Tipo de direccionamiento en ensamblador para acceder a píxeles seleccionado: Direccionamiento indexado: Para acceder a los píxeles durante el procesamiento en ensamblador, se optó por utilizar direccionamiento indexado, calculando dinámicamente las posiciones de memoria a partir de los índices correspondientes a cada píxel.

Esta elección se alineó de manera natural con la estrategia de utilizar un buffer completo en memoria para almacenar la imagen de entrada y la imagen interpolada. Al cargar toda la imagen en memoria, los datos de los píxeles se encuentran almacenados de manera continua y ordenada, lo que permite calcular sus posiciones exactas mediante operaciones aritméticas sobre los índices de fila y columna, sin necesidad de realizar operaciones de entrada/salida adicionales.

El direccionamiento indexado facilita el acceso a cualquier posición de la imagen de forma flexible y dinámica. Para cada píxel procesado, se calcula su desplazamiento dentro del buffer en función de su posición relativa, utilizando registros para realizar las operaciones de multiplicación y suma necesarias. Esta estrategia permite que el mismo código sea reutilizado para acceder a diferentes ubicaciones de la imagen, sin depender de valores constantes o de una estructura fija de memoria.

Además, el direccionamiento indexado es esencial en un contexto donde la imagen cambia de tamaño como consecuencia de la interpolación. El crecimiento de la imagen a 385×385 píxeles tras el proceso de interpolación requiere un esquema de direccionamiento adaptable, que pueda calcular de forma precisa las nuevas posiciones de los píxeles generados sin reescribir grandes bloques de código.

9) Optimización en uso de registros seleccionada: Uso combinado de registros y memoria: Durante la implementación del procesamiento de la imagen en ensamblador, se adoptó una estrategia de uso combinado de registros y memoria para optimizar el manejo de los datos de píxeles durante el proceso de interpolación bilineal.

En el desarrollo del programa, los datos más críticos, como los valores base de los píxeles (A, B, C, D) leídos del buffer de entrada, se almacenan temporalmente en registros (al, bl, cl, dl) para acelerar las operaciones aritméticas inmediatas. Esta decisión permite que los cálculos de interpolación entre píxeles se realicen de manera rápida, minimizando los accesos a memoria y aprovechando la velocidad de acceso de los registros internos del procesador.

Sin embargo, para conservar resultados intermedios de las interpolaciones parciales (como c1, g1, c2, g2), se empleó memoria auxiliar a través del buffer temp_interp. Los resultados de las primeras interpolaciones se almacenan en esta sección de memoria, lo cual permite liberar registros para otros usos sin perder la información necesaria para los cálculos de interpolaciones posteriores.

Este enfoque permite balancear eficientemente el uso de recursos: los registros se utilizan para cálculos inmediatos donde la velocidad es crítica, mientras que la memoria se usa para almacenar datos que deben conservarse entre diferentes etapas del procesamiento pero no requieren acceso constante en ciclos de ejecución muy cerrados.

A su vez, al almacenar resultados intermedios en memoria, se facilita la organización del flujo de procesamiento, ya que las direcciones de memoria actúan como referencias estables que permiten recuperar valores cuando se requieren nuevamente en el cálculo de píxeles interpolados más complejos.

B. Conclusión de la evaluación

La selección de opciones realizadas en este proyecto permite cumplir de manera efectiva con todos los requerimientos funcionales, de procesamiento y de salida establecidos en el diseño inicial. Se logró integrar de manera eficiente el procesamiento en bajo nivel mediante lenguaje ensamblador y la automatización y validación de resultados mediante herramientas de alto nivel.

Esta propuesta destaca por:

- **Automatización del flujo de trabajo:** El uso de Python permitió automatizar la preparación de la imagen de entrada, la ejecución del programa en ensamblador y la visualización de los resultados, eliminando la necesidad de intervención manual entre etapas.
- **Validación eficiente de resultados:** La visualización automática mediante script facilitó la validación inmediata de los resultados de interpolación, mejorando el tiempo de respuesta y reduciendo posibles errores humanos en la verificación.
- **Compatibilidad y facilidad de desarrollo:** La elección de Linux como sistema operativo proporcionó un entorno nativo para ensamblaje, enlazado y ejecución de programas en bajo nivel, simplificando la integración de herramientas y reduciendo problemas de compatibilidad.
- **Madurez y disponibilidad de herramientas:** La selección de la arquitectura x86 ofreció un amplio conjunto de instrucciones y soporte documental, facilitando la implementación de operaciones de manipulación de datos y el control eficiente de registros y memoria.
- **Optimización del acceso a datos:** La decisión de cargar completamente en memoria la imagen de entrada evitó múltiples operaciones de lectura/escritura, mejorando el rendimiento general y simplificando el acceso a los datos de píxeles.
- **Calidad visual en la interpolación:** El recorrido de píxeles con solapamiento permitió generar transiciones

más suaves en la imagen interpolada, asegurando una representación más continua y precisa de los valores intermedios.

- **Gestión eficiente del almacenamiento de resultados:** El uso de un buffer de memoria para la imagen interpolada permitió almacenar todos los nuevos datos de forma ordenada antes de su escritura final, adaptándose al incremento de tamaño a 385×385 píxeles.
- **Flexibilidad en el acceso a memoria:** El direccionamiento indexado posibilitó calcular dinámicamente las posiciones de los píxeles, facilitando el manejo de imágenes de diferentes tamaños y mejorando la adaptabilidad del código.
- **Balance entre rendimiento y simplicidad:** La estrategia de uso combinado de registros y memoria optimizó el procesamiento de datos críticos en registros, mientras que almacenó resultados intermedios en memoria, equilibrando la velocidad de ejecución con la organización estructurada de los datos.

En conjunto, las decisiones tomadas aseguran un diseño eficiente, coherente y alineado a los objetivos del proyecto, maximizando la calidad de la interpolación bilineal realizada en ensamblador y garantizando una integración fluida con las herramientas de automatización y visualización en alto nivel.

REFERENCES

[1] ARM Ltd., *ARM Architecture Reference Manual*, 2012.

[2] M. Tournier, F. Berry, *Optimized Image Processing Using Assembly Language*, Proceedings of the Real-Time Multimedia Systems Symposium, 2019.

[3] Conventional Commits Specification. https://www.conventionalcommits.org/en/v1.0.0/

[4] M. Barr, *Programming Embedded Systems in C and C++*, 2nd ed., O'Reilly Media, 2006.

[5] Vincent Driessen, A successful Git branching model. https://nvie.com/posts/a-successful-git-branching-model/

[6] IEEE Computer Society, *IEEE Recommended Practice for Software Requirements Specifications*, IEEE Std 830-1998, 1998.

[7] Intel Corporation, *Intel 64 and IA-32 Architectures Optimization Reference Manual*, 2023. Disponible en: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

[8] ISO/IEC 9126: Software Engineering — Product Quality. https://iso.org/standard/22749.html

[9] ISO/IEC 25010: Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models. https://iso.org/standard/35733.html

[10] John Gruber, Markdown: Syntax. https://daringfireball.net/projects/markdown/syntax

[11] Massachusetts Institute of Technology, *Introduction to Computer Systems Engineering (6.111)*, MIT OpenCourseWare, 2020. Disponible en: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-111-introduction-to-computer-systems-engineering-spring-2020/

[12] OpenCV Developers, *Open Source Computer Vision Library*, OpenCV.org, 2024. Disponible en: https://opencv.org y documentación técnica en https://docs.opencv.org/4.x/d3/d63/classcv_1_1Mat.html

[13] OpenCV, *Geometric Transformations of Images*, OpenCV Documentation, 2024. Disponible en: https://docs.opencv.org/4.x/da/d6e/tutorial_py_geometric_transformations.html

[14] RISC-V Foundation, *The RISC-V Instruction Set Manual*, 2019.

[15] Intel Corporation, *Intel 64 and IA-32 Architectures Software Developer's Manual*, 2022.